```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import time
from scipy import optimize

# Paramètres du bras robotique
q = np.array([-80, -80, -20, 0])  # En degrés
l = np.array([2, 1, 1, 0.5])  # Longueur des segments du robot

# Ouvrir le flux vidéo de la caméra
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    print("Erreur : Impossible d'ouvrir la caméra")
    exit()

lower_skin = np.array([50, 100, 50], dtype=np.uint8)  # Plage basse du vert
upper_skin = np.array([90, 255, 255], dtype=np.uint8)  # Plage haute du vert

# Variables pour suivre la position de la main
previous_position = None
previous_time = time.time()

# Paramètres de mise à l'échelle pour le bras robotique
# La mise à l'échelle va convertir la position de la caméra (en pixels) vers le système du bras robotique.
# Limites de la fenêtre de la caméra (en pixels)
CAMERA_WIDTH = 640
CAMERA_HEIGHT = 480

# Limites de la zone de travail du bras robotique (en mètres)
ROBOT_WORKING_AREA_X = 2.5  # par exemple 3 mètres
ROBOT_WORKING_AREA_Y = 3.5  # par exemple 3 mètres

# Fonction de transformation pour le bras robotique
def rotz(theta):
    return np.array([[np.cos(theta), -np.sin(theta)],
                     [np.sin(theta), np.cos(theta)]])

def homogeneousMatrix(theta, tx, ty):
    R = rotz(theta)
    T = np.array([[tx], [ty]])
    ligne3 = np.array([[0, 0, 1]])
    return np.vstack((np.hstack((R, T)), ligne3))

def poseEffecteur(q, l):
    """ Calcule la position des articulations du bras """
    c0 = np.array([[0], [0], [1]])
```

```
    wMso = homogeneousMatrix(np.pi / 2, 0, 0)
    soMs1 = homogeneousMatrix(np.radians(q[0]), l[0], 0)
    wMs1 = wMso @ soMs1
    c1 = wMs1 @ c0
    s1Ms2 = homogeneousMatrix(np.radians(q[1]), l[1], 0)
    wMs2 = wMs1 @ s1Ms2
    c2 = wMs2 @ c0
    s2Ms3 = homogeneousMatrix(np.radians(q[2]), l[2], 0)
    wMs3 = wMs2 @ s2Ms3
    c3 = wMs3 @ c0
    s3Me = homogeneousMatrix(np.radians(q[3]), l[3], 0)
    wMe = wMs3 @ s3Me
    c4 = wMe @ c0
    return c0, c1, c2, c3, c4

#pose init
c0, c1, c2, c3, c4 = poseEffecteur(q, l)
ce = np.copy(c4)
ce_target = np.copy(ce)

def erreurPosition(q, ce, l):
    """ Fonction d'optimisation pour minimiser l'écart entre le bras et la position cible """
    _, _, _, _, c4 = poseEffecteur(q, l)
    return np.linalg.norm(c4 - ce)

# Création de la figure avec 2 sous-graphiques
fig, ax2 = plt.subplots(1, 1, figsize=(12, 6))

# Graphique du bras robotique
ax2.set_xlabel("X")
ax2.set_ylabel("Y")
ax2.grid()

# Définir les limites pour centrer le graphique avec x=-0.25 au coin bas-gauche
# Cela va laisser de l'espace pour le corps du robot et le placer dans le coin en bas à
gauche
ax2.set_xlim(-0.25, 2.75)  # X allant de -0.25 (coin) à 3
ax2.set_ylim(0, 3.75)      # Y allant de 0 à 3 (pour que l'origine soit en bas)

# Liste pour enregistrer les positions de la pince (dernière articulation du bras)
pince_positions = []

# Boucle principale
while True:
    ret, image = cap.read()
    if not ret:
        print("Erreur")
        break
```

```
    # Conversion en HSV et détection des contours de la main
    hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsv, lower_skin, upper_skin)
    mask = cv2.medianBlur(mask, 5)
    mask = cv2.erode(mask, None, iterations=2)
    mask = cv2.dilate(mask, None, iterations=2)
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


    if contours:
        max_contour = max(contours, key=cv2.contourArea)
        if cv2.contourArea(max_contour) > 1000:
            M = cv2.moments(max_contour)
            if M["m00"] != 0:
                cx = int(M["m10"] / M["m00"])
                cy = int(M["m01"] / M["m00"])

                # Inverser l'axe Y pour l'image (pour l'affichage avec l'origine en bas à gauche)
                #inverted_cy = CAMERA_HEIGHT - cy

                # Dessin du point central et des contours de la main
                cv2.drawContours(image, [max_contour], -1, (0, 255, 0), 2)  # Dessiner les
contours
                cv2.circle(image, (cx, cy), 7, (255, 0, 0), -1)  # Dessiner le point central sans
inversion

                # Mise à l'échelle de la position détectée en pixels vers les coordonnées du robot
                scale_x = ROBOT_WORKING_AREA_X / CAMERA_WIDTH
                scale_y = ROBOT_WORKING_AREA_Y / CAMERA_HEIGHT

                # Coordonnées du robot dans l'espace de travail
                robot_x = cx * scale_x
                #robot_y = inverted_cy * scale_y
                robot_y = cy * scale_y

                # Calcul des nouveaux angles pour atteindre la position de la main

                ce_target = np.array([[robot_x], [robot_y], [1]])
                alpha = 0.3 # Facteur d'interpolation
                ce = ce + alpha * (ce_target - ce)
                qnext = optimize.minimize(erreurPosition, q, args=(ce, l))
                c0, c1, c2, c3, c4 = poseEffecteur(qnext.x, l)

                # Ajouter la position de la pince à l'historique (c4)
                pince_positions.append([c4[0, 0], c4[1, 0]])
```

```
            # Mise à jour du graphique du bras robotique
            ax2.clear()
            ax2.set_xlabel("X")
            ax2.set_ylabel("Y")
            ax2.grid()

            # Re-définir les limites pour que le bras ait plus d'espace
            ax2.set_xlim(-0.25, 2.75)  # Ajustement de l'axe X pour centrer à -0.25
            ax2.set_ylim(0, 3.75)      # Ajustement de l'axe Y pour un affichage propre

            # Tracer l'historique de la position de la pince en rouge
            pince_positions_array = np.array(pince_positions)
            ax2.plot(pince_positions_array[:, 0], pince_positions_array[:, 1], 'r-', linewidth=1)

            # Tracer la position actuelle du bras
            ax2.plot([c0[0, 0], c1[0, 0], c2[0, 0], c3[0, 0], c4[0, 0]],
                     [c0[1, 0], c1[1, 0], c2[1, 0], c3[1, 0], c4[1, 0]], 'g-', linewidth=2)
            ax2.plot([c0[0, 0], c1[0, 0], c2[0, 0], c3[0, 0], c4[0, 0]],
                     [c0[1, 0], c1[1, 0], c2[1, 0], c3[1, 0], c4[1, 0]], 'go', markersize=8)

            # Mise à jour des affichages
            plt.pause(0.01)

    # Affichage de la caméra
    cv2.imshow("Main detectee", image)
    cv2.imshow("Masque peau", mask)

    # Quitter avec 'q'
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Fermeture propre
cap.release()
cv2.destroyAllWindows()
plt.close(fig)
```